# VZ LINK magazine

ISSUE 56

JUNE 1991

NEW ZEALAND

VZ 300/200
COMPUTER USERS CLUB

# ODDS AND ENDS

Hi, Well we finally managed to get out the JUNE issue of your VZ newsletter.
I am sure you have all been looking forward to your monthly shot of VZ news and programmes. There never seems to be enough hours in the day to complete everything one needs to get finished, and this seems more noticeable with computer programming.

I am still hoping that we get a good turnout at the next club meeting. The details are below. It is always good to see the members and a large flow of information seems to take place at these meetings.

## REMEMBER

If you want to contact me about any problems the phone number is now: Ph. 520-6743
I have the answer phone connected so you can ring any time and leave a message,
I will ring you back. Please leave your number if it has changed.

## ACTIVITY EVENINGS.

The last few activity nights were lonely ones, with only Ken and myself attending.
Therefore I will scrap that idea for the time being. Maybe the wet and cold nights have put most people off. I will look at starting them again in the summer.
If you have problems etc. then you are most welcome to call around.
But please ring me first, as I maybe away or on call.

## MEETING PLACE REQUIRED

I am currently looking around for a new place where we can hold our meetings. I would prefer somewhere central and free of charge if possible. If you know of a hall etc that may fit the bill, please let me know.

## QUERY ON WORD PROCESSORS

Ken Clarke has pointed out to me that anybody Customising the E&F WORD PROCESSOR as per the item by Bob Kitch in the October 90 issue of VZ LINK (issue 48) may have problems when it comes to loading files on other systems with different memory configurations. If you have done this mod. could you please contact me.

## CLUB MEETING

I am sorry we had to cancel the last club meeting, due to vandals cutting the power to the hall.
The club meeting will be now be held on :-

>>>>>>>>>>>>>>>>>>>>  SATURDAY 13 JULY 1991  <<<<<<<<<<<<<<<<<<<<<<<<
>>>>>>>>>>>>>>>>>>>>  Starting at 10.00 am.  <<<<<<<<<<<<<<<<<<<<<<<<

Don't forget to bring a plate and your problems and ideas.

See you next issue.

---

P.O BOX 1972 C.P.O AUCKLAND N.Z
AUCKLAND VZ 300/200 COMPUTER USERS CLUB (1986)
NEW ZEALAND VZ 300/200 COMPUTER USERS CLUB (1989)

ALL MATERIAL IN THIS NEWSLETTER IS SUBJECT TO COPYRIGHT.
CONTRIBUTED PROGRAMS AND ARTICLES CAN BE REPRODUCED ON THE UNDERSTANDING THAT IT IS FOR THE PRIVATE USE OF SUBSCRIBERS ONLY.

COPYRIGHT IS ALWAYS RETAINED BY THE AUTHOR.
THIS CLUB WILL NOT BE HELD RESPONSIBLE FOR ANY BREACHING OF COPYRIGHT CAUSED BY THE PUBLICATION OR USE OF ANY MATERIAL SENT IN UNDER A FALSE NAME.

YOU ARE WELCOME TO PHONE ME ON.....PHONE 520-6743

---

# LIVEN-UP ANIMATION & GRAPHICS

by Bob Kitch.
7 Eurella St.,
KENMORE QLD. 4069

Want to have a utility that displays high speed animation and graphics without flicker? Then this is the project for you! Read on!

This article is the most comprehensive and integrated software project for the VZ ever attempted in a Users Group magazine. The project consists of five programs - three written in BASIC (SET-UP, LOAD-UP and THROW-UP) and two written in Assembler (START-UP and MOVE-UP) - resulting in a package that will "LIVEN-UP" the use of the VZ. The group of programs provide a utility in, what I term, FAST BASIC. Assembler is used to speed up functions that are too slow in BASIC. This most often occurs in two instances - the first is when high-speed processing is required, and the second is when precise control of a peripheral is required. Each Assembler program fulfils one of these shortcomings of BASIC. (MOVE-UP and START-UP respectively). Nine Assembler "primitives" for screen moves, graphics or animation are provided in MOVE-UP. They provide an intermediate introduction to Assembly Language programming.

For those who can't wait for the entire group of listings to be published over the coming months, (or can't be bothered keying them in) I am willing to provide the entire project on disk for $20. This includes all source and object code plus a set of picture files.

BACKGROUND TO PROJECT.

For some time now I have been contemplating some real-time hi-res graphics displays on the VZ to liven-up interest in the computer. Also, in a previous article, (see LE'VZ #24, pg. 3 & 4) I have provided details of hi-speed screen move subroutines. Furthermore, the additional memory capacity of the 64K memory expansion, that some Users have, has not been greatly utilized by programmers. This article, accompanied by the suite of programs, provides some insight into the three aspects just detailed. The programs allow a PICTURE SHOW type of display to be implemented without constant up-loading from disk. The programs are written for VZ's that have 64K memory expansion packs and are disk-based - although the programs could be easily adapted to VZ's having 16K expansions and tape units.

For those interested, the software packages used to develop this utility were as follows -

-standard BASIC and DOS ROMS.

-the EDASM used was DSE's model patched with DISKOPS4.

-QUICKWRITE Text Ed allows DISKOPS source files to be handled as W-files.

-the Monitor used was from DISK DOCTOR and was used for patching and breakpointing.

-picture files were built using ART GALLERY.

DESIGN CONCEPT.

A hi-res screen on the VZ occupies 2048 bytes (2K) of memory. The 64K memory expansion module contains four by 16K banks of RAM memory. When programming in BASIC, only memory banks 0 and 1 are easily accessed. Assembly Language programmers can access banks 2 and 3, which, in fact, overlap in address space with bank 1. A form of

"memory bank switching" is used to overcome this overlay problem.

When programming in BASIC, the Program Statement Table and Variable List Table both reside in lo-mem (generally in bank 0 although large programs extend into bank 1). The Systems Area, consisting of the Video Display Area and Communications Area, also reside in bank 0. The DOS Vector, String Area and Stack are all organised, on boot-up, into hi-mem located at the top of bank 1. The intervening area of RAM is Free Space, and as program developement and execution proceed, the various tables (with some exceptions) dynamically build into this Free Space. Switching banks 1/2/3, when in BASIC, normally results in a crash, as the hi-mem tables become "dismembered" from the lo-mem tables.

It is possible to reorganize and restrict the "BASIC Work Area" entirely into bank 0 - provided that the programs and their associated dynamic and static tables are kept short. Memory banks 1 to 3 are then "Reserved" for other use such as video screen buffer areas. The BASIC work area extends from 7AE9H to BFFFH. The problem of switching banks whilst running BASIC programs becomes trivial and hi-speed moves from hi-mem to the Video RAM area become possible by short Machine Language calls. It is also necessary for the BASIC program to be able to pass variables to the M/L routine - but this is also easy to achieve. (the Load Map for the project is provided in the remarks on the end of SETUP)

The top of bank 0 is BFFFH. Banks 1 to 3 occupy C000H to FFFFH. It is possible to buffer eight 2K hi-res screens in each memory bank, making a total of 24 screens that can be stored in the 64K memory pack. The powerful Z80 Block Move routine can move 2K of memory around in 12.16 msec. (that's fast!) with the Z80 running at 3.58 MHz. The screen display is updated every 20 msec. Alternatively, some 96 lo-res screens (512 bytes) could be stored internally. Obviously loading the 48K of data into the memory is best achieved on a disk-based system. Routines to handle the loading of data and the shifting of data between hi- and lo- mem (both directions) are required.

The ability to move data around in RAM at this speed allows animated graphics to become a possibility. The technique implied is not real-time stuff however. The method really uses "pre-formatted" screens or "pages" that are paged into the video display area.

I am sure everyone who has used the VZ in hi-res mode will have noticed the annoying "flicker" on the screen whenever an image is updated. The cause of this effect is the subject of a further article. It is sufficient to say here, that the flicker occurs whenever there is a conflict between the timing of a Z80 write operation and a 6847 VDG read operation. By intercepting the Interrupt signal (previously explained in LE'VZ #23, pg. 10 & 12 and LE'VZ #24, pg. 8 & 9) and using a "screen buffer area", it is possible to overcome the "screen hash" problem.

PROGRAM DESCRIPTION.

It was decided that three BASIC subprograms could achieve the desired effect and keep program length short.

The first BASIC program is called SETUP and displays a series of introductory screens to the user. SETUP then calls the first of the Assembler routines called STARTUP.

STARTUP carries out a number of functions that are more easily achieved from a low level language. As this routine is only needed temporarily, it is loaded into part of the hi-res video RAM area. First the TOM is lowered to BFFFH and all BASIC pointers are reset accordingly. Next, the 310 byte DOS vector is placed below the new TOM by jumping to 4004H in the DOS ROM. This invokes a reboot of the system and the READY message will appear. Control is passed back to the BASIC ROM and the DOS Vector at 79ACH called. To autorun the next BASIC program called LOADUP requires a smart bit of programming and knowledge of the DOS ROM. The source code for STARTUP details all. LOADUP is booted by STARTUP.

The second BASIC program called LOADUP is run next. A number of checks on the configuration of the VZ system are carried out to confirm that all is O.K. The bank switching of the 64K pack is exercised and verified as present and operational. Next, the first 4 identification bytes (AAH, 55H, E7H, 18H) of the DOS ROM are verified to confirm that a disk-based system is in place. The boot-up of the system, carried out by the Initial Program Loader, seeks these four bytes at 4000H, 6000H and 8000H to learn the configuration of the sytem being booted. Any ROM cartridge inserted at these locations needs these identification bytes to be recognized. After this, the top-of-memory location under the DOS vector is checked so that no conflict across the bank 0 to bank 1/2/3 address can occur.

LOADUP then further lowers the top-of-memory to AFFFH (the DOS vector is not moved). This creates an area of "protected memory". The second of the Assembler routines called MOVEUP is loaded in to this area of memory. A small M/L routine is also loaded into protected memory. Fourteen bytes of the familiar Block Move routine are loaded in and the USR vector set to point at the start of this routine. The routine as loaded has the HL register (Source) set to the start of the Video Display RAM at 7000H and which extends to 77FFH (The 2K screen area). The DE register (Destination) is first set to the bottom of bank 1/2/3 at C000H. The Size or BC register is set to 800H or 2K. As the 2K screen pages occur on simple boundaries in address space, (C000H, C8000H, D000H, D8000H etc.) it is only necessary to alter the Most Significant Byte of the Destination address as the 2K screen files are loaded from disk. The address of this byte in the M/L routine is assigned in line 1800 to enable the BASIC program to pass variables into the M/L routine.

It was decided that the easiest way to present the 24 filenames for the screens to be presented to LOADUP was via DATA statements located in lines 2000 to 2080. There is scope to alter this if required. The screens are built using a graphics editor. I would suggest Bruce Kitch's ART GALLERY as being suitable. The files must load into the Video Display area i.e 7000H-77FFH. (but see later comments).

Having come this far, the actual read-in loop is quite simple! A triple-nested loop construct is used. The outer loop switches the memory banks sequentially. The intermediate loop passes the MSB of the Destination in the selected bank to the M/L routine. The inner loop reads the filename from the data statement, checks its size, pokes the name into another area of the program and finally loads the file into the Video Display Area. Finally the screen page is moved up into the memory bank for storage. This completes the work that LOADUP carries out.

```
0001 ;
0002 ;SOURCE:START
0003 ;ORIGIN:7200H (VRAM).
0004 ;OBJECT:STARTUP
0005 ;
0006 ;THIS ROUTINE:-
0007 ;1. LOWERS TOM TO "TOP"
0008 ;2. RESETS POINTERS BY A NEW
0009 ;3. RELOCATES DOS VECTOR TO
0010 ;   BELOW "TOP".
0011 ;4. RUNS "FNAM" (T:FILE).
0012 ;
0013 ;
0014 ;
0015 TOP  EQU  0BFFFH
0016 ;VALUE FOR TOM.
0017 CLS  EQU  01C9H
0018 ;CLEAR SCREEN.
0019 NEW  EQU  1B4DH
0020 ;NEW ENTRY POINT.
0021 TOM  EQU  78B1H
0022 ;TOM POINTER.
0023 JUMP EQU  0C3H
0024 ;OPCODE FOR JUMP.
0025 TURN EQU  0C9H
0026 ;OPCODE FOR RETURN.
0027 RDOS EQU  4004H
0028 ;ENTRY POINT TO RESET DOS
0029 ;VECTOR.
0030 PWON EQU  4008H
0031 ;ENTRY POINT FOR DISK POWER
0032 ;ON.
0033 READ EQU  45DBH
0034 ;ENTRY POINT TO READ FILNAM.
0035 VEC2 EQU  79ACH
0036 ;VECTOR CALLED FROM 1A1CH.
0037 ;
0038 ;
0039 ;
0040      CALL CLS
0041      LD   HL,TOP
0042      LD   (TOM),HL
0043      LD   HL,ENT
0044      LD   (VEC2+1),HL
0045      LD   A,JUMP
0046      LD   (VEC2),A
0047      JP   RDOS
0048 ;
0049 ;
0050 ENT  POP  HL
0051      LD   A,TURN
0052      LD   (VEC2),A
0053      EI
0054      CALL NEW
0055      CALL PWON
0056      DI
0057      LD   HL,FNAM
0058      JP   READ
0059 FNAM EQU  $
0060 *"LOADUP":*
0061 END  EQU  $
```

The final BASIC program called THROWUP is autoloaded. This is a simple routine that uses a triple-nested loop similar to that outlined for LOADUP except that the screen pages are brought down from hi-mem to the Video Display Area. The MSB of the Source and Destination addresses are interchanged to achieve this change in sense of movement. The inner loop is a delay loop to slow down the speed of the paging. As written, two options exist to display the 24 screens. A simple or plain display uses the small Block Move routine. A considerable amount of "hash" appears using this non-interrupt driven routine. The screen buffer is not used. The second option for display uses the routines in MOVEUP. These use the interrupt-driven display system and are completely "hash-free".

DISCUSSION OF MOVEUP.

The source code for MOVEUP is by far the most interesting and illuminating portion of the project. It consists of a number of general purpose "primitive" subroutines.

There are nine "fancy" screen move routines. The action of these is described by their titles - splat, open-up, roll up/down, push up/down, four bar roll down, and l-r/r-l sweeps. With the exception of the sweeps, all other screen moves involve vertical movements of the pictures. This movement is the easiest to program as consecutive blocks or lines of the screen are moved using the Block Move command. Most of the additional programming keeps track of the appropriate source, destination and size pointers. Usually either a 2K block (one full screen), or a 32 byte (one screen line), portion of memory is required to be moved.

The sweeps are much more difficult to program. The logic used to achieve the horizontal replacement, two bits at a time, is the subject of a further article on logical operators. Experienced programmers may follow the method through the comments inserted in the source code. Some very elegant mathematics is used.

All of the move routines write to a screen buffer rather than directly into the video display RAM area. Also during this write period, all interrupts are disabled so that processing speed is optimized. The screen buffer is used so that the video display may be interrupt driven such that flicker or hash is avoided.

The timing of the interrupt driven display, that eliminates the screen hash, is achieved by two routines that are called from the Interrupt Vector. The screen buffer is moved in two halves as there is insufficient time in the display flyback period to move the entire 2K screen. A display and pause routine is entered to permit the screen buffer to be moved into the video display RAM. The pause loop is located in the DOS ROM, and by choosing a suitable duration, a smooth movement can be obtained.

Two other general purpose subroutines are given that save and restore the register set to the stack upon entry and exit to any machine code. Furthermore, the register values are unchanged. Very useful.

Finally, the use of a Jump Table to enter the nine routines considerably simplifies the set-up of the USR pointers. This technique is to be recommended.

FURTHER ENHANCEMENTS.

That about outlines the essentials of the programs. Note that instructions are inserted in the code to allow the procedures to be followed. Also the code has been written with specific constants inserted rather than generalized variables. For those who are interested, it is possible to store a couple of further screens in the top of bank 0 by setting the top-of-memory even lower but the elegance of the triple-nested loops is lost! (refer Load Map).

A further refinement of the loading and saving technique is shown in the code for THROWUP. On reflection and when the 24 screens are loaded into the buffers in hi-mem, there are in reality three 16K blocks of data extending from C000H to FFFFH. It would be convenient and a little faster, to save and load these 16K blocks to- and from-disk respectively. Dialog encountered during program execution indicates how this refinement can be achieved.

Observant programmers will notice that subroutine SC0 "(Splat") in MOVEUP is not used in the present version. This can be used in lieu of the 14 byte M/L routine and provide hash-free animation.

One further enhancement is to use data compression techniques to store the screens into the buffer areas. Most graphic screens have many sections of "shared" bytes, for example, in "background" areas. This would be a very fruitful area for experimentation as many more than 24 screens could be stored in the 64K pack.

I trust that this article provides some insight into an interesting area of programming. Should anyone wish to communicate with me on their efforts, then I would be pleased to do so. A number of these enhancements are operational on my "fully-fledged" version.

---

A potpourri of ideas, software modifications, help-line, your ideas, suggestions, gripes, where to find it, cheats, pokes and infinite lives, adventure gaming help-line, hardware help in construction, or in fact anything that other VZ'ers could be interested in or be able to find help for you.....

*It is HANDS ON THE THE VZ JOYSTICKS this month, with an up-date from David Lambourn to his article which appeared in VZ-LINK #40 [Feb. '90]. Careful examination will show that using the figure of 34 as per the example, ties the ARM & FIRE function on one 'STICK together, and yet on the other, leaving the ARM function floating. Change the number to 32 and both 'STICKS become identical with ARM & FIRE working as one function - both returning the figure 15 when either are operated. For further enhancement, if the number is changed to 42 - BOTH 'STICKS then separate the ARM & FIRE functions. Both returning the figure 15 when FIRE is operated, and 30 when ARM is operated. [See my program of 'DONKEY KONG', which illustrates using the ARM & FIRE buttons as JUMP LEFT & JUMP RIGHT and the 'STICK for MOVING LEFT & MOVING RIGHT (without jumping) as well as UP the ladders].*
*** NEED ANY HELP 'JOYSTICKING' A PROGRAM? THEN WHY NOT ASK? ***

# MENU PROGRAMME

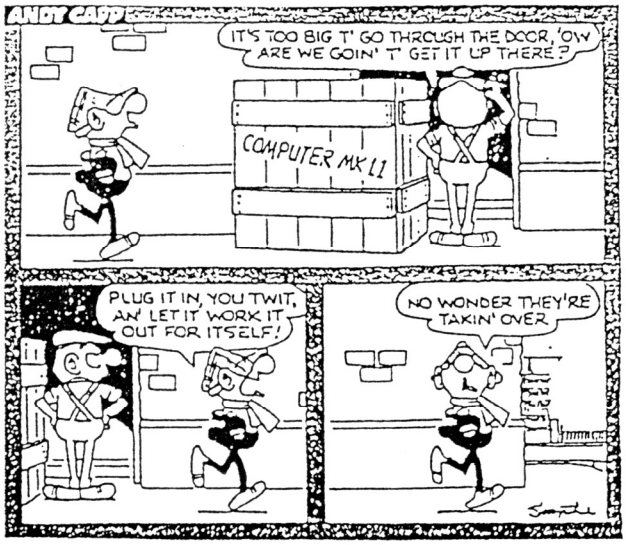
```
5 'GENERAL PURPOSE MENU FOR CALLING UP TO 24 FILES.     22.12.86
8 POKE 30744,0:COLOR,0
10 CLS:PRINT:PRINT"  [illegible]"
20 PRINT @ 96," <A> DEPTH CHARGE GAME            ";
30 PRINT:PRINT" <B> OUTLAW - FIVE DESPARADOS    ";
40 PRINT:PRINT" <C> DONKEY KONG GRABS A MAIDEN ";
50 PRINT:PRINT" <D> SATURN - JOURNEY INTO SPACE";
60 PRINT:PRINT" <E> PEDRO THE FISHERMAN         ";
70 PRINT:PRINT" <F> ST GEORGE AND THE DRAGON    ";
80 FR%=1:GOSUB 1100:'GET OPTION
120 PRINT @ 96," <G> JUKEBOX FAVOURITES - TOP 12";
130 PRINT:PRINT" <H> ROTARY WANKEL MODELLING     ";
140 PRINT:PRINT" <I> RESISTOR COLOUR CODES       ";
150 PRINT:PRINT" <J> SOUND EFFECTS               ";
160 PRINT:PRINT" <K> TEAPOT SONG                 ";
170 PRINT:PRINT" <L> GLADIATORS IN COMBAT        ";
180 FR%=2:GOSUB 1100:'GET OPTION
220 PRINT @ 96," <M> TEDDYBEARS' PICNIC          ";
230 PRINT:PRINT" <N> RABBIT BREEDING             ";
240 PRINT:PRINT" <O> HAPPY BIRTHDAY              ";
250 PRINT:PRINT" <P> XXXXXXXXXXX                 ";
260 PRINT:PRINT" <Q> XXXXXXXXXXX                 ";
270 PRINT:PRINT" <R> XXXXXXXXXXX                 ";
275 FR%=3:GOSUB 1100:'GET OPTION
280 PRINT @ 96," <S> XXXXXXXXXXX                 ";
290 PRINT:PRINT" <T> XXXXXXXXXXX                 ";
300 PRINT:PRINT" <U> XXXXXXXXXXX                 ";
310 PRINT:PRINT" <V> XXXXXXXXXXX                 ";
320 PRINT:PRINT" <W> XXXXXXXXXXX                 ";
330 PRINT:PRINT" <X> XXXXXXXXXXX                 ";
340 FR%=4:GOSUB 1100:'GET OPTION
500 GOTO20
1100 PRINT:PRINT"ENTER OPTION ?  <Z> NEXT SCREEN";
1110 A$=INKEY$:A$=INKEY$
1120 IF A$="" THEN GOTO 1110
1130 IF A$="Z" THEN RETURN
1140 ST%=0:GOSUB 2000:IF ST%=0 GOTO 1110
1150 FOR V%=ST% TO ST%+31
1160 I%=PEEK(V%)
1170 IF I%<64 THEN I%=I%+64:POKE(V%),I%:GOTO 1190
1180 IF I%>63 AND I%<>96 THEN I%=I%-64:POKE(V%),I%
1190 NEXT V%:SOUND 31,3
1200 IF A$="A"THEN:RUN"DEPCHARG"
1210 IF A$="B"THEN:RUN"OUTLAW"
1220 IF A$="C"THEN:RUN"KONG"
1230 IF A$="D"THEN:RUN"SATURN"
1240 IF A$="E"THEN:RUN"PEDRO"
1250 IF A$="F"THEN:RUN"GEORGIE"
1260 IF A$="G"THEN:RUN"JUKE BOX"
1270 IF A$="H"THEN:RUN"ROTARY"
1280 IF A$="I"THEN:RUN"RESISTOR"
1290 IF A$="J"THEN:RUN"SOUNDS"
1300 IF A$="K"THEN:RUN"TEAPOT"
1310 IF A$="L"THEN:RUN"GLADIATO"
1320 IF A$="M"THEN:RUN"TEDDY"
1330 IF A$="N"THEN:RUN"RABBITS"
1340 IF A$="O"THEN:RUN"BIRTHDAY"
1350 IF A$="P"THEN:RUN"GO"
1360 IF A$="Q"THEN:RUN"GO"
1370 IF A$="R"THEN:RUN"GO"
```

```
1380 IF A$="S"THEN:RUN"GO"
1390 IF A$="T"THEN:RUN"GO"
1400 IF A$="U"THEN:RUN"GO"
1410 IF A$="V"THEN:RUN"GO"
1420 IF A$="W"THEN:RUN"GO"
1430 IF A$="X"THEN:RUN"GO"
1580 GOTO1110
2000 IF(A$="A" AND FR%=1)OR(A$="G" AND FR%=2) THEN ST%=28768
2010 IF(A$="B" AND FR%=1)OR(A$="H" AND FR%=2) THEN ST%=28832
2020 IF(A$="C" AND FR%=1)OR(A$="I" AND FR%=2) THEN ST%=28896
2030 IF(A$="D" AND FR%=1)OR(A$="J" AND FR%=2) THEN ST%=28960
2040 IF(A$="E" AND FR%=1)OR(A$="K" AND FR%=2) THEN ST%=29024
2050 IF(A$="F" AND FR%=1)OR(A$="L" AND FR%=2) THEN ST%=29088
2060 IF(A$="M" AND FR%=3)OR(A$="S" AND FR%=4) THEN ST%=28768
2070 IF(A$="N" AND FR%=3)OR(A$="T" AND FR%=4) THEN ST%=28832
2080 IF(A$="O" AND FR%=3)OR(A$="U" AND FR%=4) THEN ST%=28896
2090 IF(A$="P" AND FR%=3)OR(A$="V" AND FR%=4) THEN ST%=28960
3000 IF(A$="Q" AND FR%=3)OR(A$="W" AND FR%=4) THEN ST%=29024
3010 IF(A$="R" AND FR%=3)OR(A$="X" AND FR%=4) THEN ST%=29088
4020 RETURN
5000 STOP
10000 CLS:PRINT@106,"ERASING GO":ERA"GO"
10010 PRINT@106," SAVING GO":SAVE"GO"
10020 CLS
10030 END
```

This is a Menu with a difference. It displays the programme you want with more information, to remind you what it is all about.
In fact, up to 27 characters. [See the example on the print out above.]

Enter the details you want to display in lines 20 to 330. Enter the actual programme names in lines 1200 to 1430. Make certain that the unused lines are blanked out as is shown in lines 330 and also 1430.

Save this programme and the programmes it runs on the one disk and you will wonder just how you ever got along without it.
Note the routine at 10,000.
Use <GOTO 10010> to SAVE to a new disk.
<GOTO 10000> to REPLACE after adding amendments.



# SPEEDING UP LOOPS IN BASIC

by Pete Patterson

Loops in a computer program are probably the most useful task a computer does,but other than being used in deliberate time delays,they can slow a program down quite markedly. Therefore when writing a program in BASIC you should do everything you can to make the loops run as fast as possible. The FOR-NEXT loop is the best and fastest type of loop to control things in BASIC. As an example,compare a countdown loop to a FOR-NEXT loop,both doing a Ten Thousand count.

```
10 X=10000
20 X=X-1
30 IFX>0THEN20
40 SOUND30,1
```

This program takes 1 minute 55 seconds to execute. Pretty yuck compared to the FOR-NEXT loop.

```
10 FORX=1TO10000
20 NEXTX
30 SOUND30,1
```

Run this simple loop and you will find that it takes about 24 seconds to execute. That is certainly a lot faster than the countdown loop but as not as fast as it could be.

BASIC is an interpretive language,meaning it must read and evaluate each character that you type. Some of its operations are implicit,meaning that it will do the right thing whether you have the "right" thing in your program or not.

A perfect case in point is the NEXT statement in the FOR-NEXT loop. You can speed up the program dramatically by simply eliminating the X in line 20. This drops the execution time of the loop to about 18 seconds.

Why should one character make such a difference? It is all in the way BASIC does things. First,when you say FORX=1TO10000,BASIC sets up a counter in memory. NEXT is the trigger that tells the computer to add one more to the counter and check to see if it has satisfied the requirement by having completed the count. When you have an X (or any variable name) on the NEXT line,the computer calls in another routine which simply locates variables by name in memory. Easy enough,but when you do it 10,000 times,it adds up.

Okay,we can still get this loop to run faster. Add this line to the program and run it again.

For VZ's with Extended BASIC

```
5 DEFINTX
```

For VZ's without Extended BASIC

```
5#X
POKE31469,153
```

The execution time with the X in line 20 is down to about 17 seconds and with the X removed to about 11 seconds. What's the difference? BASIC automatically assumes that all numbers are floating point or double precision numbers. That means that all calculations are carried out to 16 places and can have a range of 10 to the power of 62 to 10 to the power of -64. That is a lot of overhead for just counting from 1 to 10,000(or in other situations where you don't need decimal places). The Z80 microprocessor inside the VZ can handle integer addition a lot faster than floating point. DEFINT (DEF)ines (INT)egers,we declared X to be an Integer,which in BASIC means it is a whole number(no decimal places) between -32768 and +32767. (Refer Bob Kitches article in VZ LINK Issue 55).

You can count only up to 32767 if you start at 1,but if you need a larger faster loop,you can start at negative values all the way down to -32768 and thus extend you looping range to 65,000.

If in the extreme case,you need to do something more than 65,000 times,it is still faster to nest two Integer loops (put one inside the other),than to use a floating number to control the loop.

Take an example:

```
10 DEFINTX,Y   or 10#X,Y
                  POKE31469,153
20 FORX=1TO100
30 FORY=1TO100
40 NEXT:NEXT
50 SOUND30,1
```

When you run this program you will find that it takes only a second longer to count to 10,000 with the two loops. Using NEXTX and NEXTY will cost the same six second penalty as it did with the earlier program.

Try doing a double loop for a hundred thousand:

```
10 DEFINITX,Y
20 FORX=1TO1000
30 FORY=1TO100
40 NEXT:NEXT
50 SOUND30.1
```

It takes 1 minute 52 seconds. Not bad for something that is done a hundred thousand times. Thats about the same time it took the countdown loop to do its thing ten thousand times.

By the way adding DEFINTX to the countdown loop brings its execution time down to 1 minute 43 seconds. Still pretty yuck.

---

## QUICK RECIPES FOR COMPUTER USERS

This is another idea which seems like it maybe fun. Send in your quick recipes for dishes which you can whip together while the computer does it's thing (sorting etc.)

### MINCE HASH  from Brenda

Cooking time 45 minutes.
( about the time it takes a certain unnamed database to sort data )

500g Steak Mince.
400g Tin Spaghetti in tomato sauce.
310g Tin Kernel Corn
2 to 3 Onions (chopped)
2 Carrots (sliced)
2 Beef oxo cubes
Half Cup grated Cheese
Quarter Cup frozen Peas
1 tbsp Salad dressing mix or mustard
1 1/4 Cup Water

Brown Mince and Onions for 5 minutes. Add Carrots and Water to Mince & Onions.
Mix Oxo and Salad dressing to a paste and add to meat mixture and simmer for 35 minutes.
Add rest if ingredients and simmer for further 5 minutes.
Serve with with fresh bread or rolls.

Will feed 2 Adults and 2 Children.

PROPORTION OF TUBES

```
10 "        SPECIFICATIONS FROM AN IRISH GOVERNMENT DEPARTMENT
20 "
25 "    1.  ALL PIPE IS TO BE MADE OF A LONG HOLE, SURROUNDED BY
26 "        METAL OR PLASTIC ROUND THE HOLE.
27 "
30 "    2.  ALL PIPE IS TO BE HOLLOW THROUGHOUT THE ENTIRE LENGTH
40 "        DO NOT USE HOLES OF DIFFERENT LENGTH THAN THE PIPE.
45 "
50 "    3.  THE I.D.(INSIDE DIAMETER) OF ALL PIPE MUST NOT EXCEED
60 "        THE O.D.(OUTSIDE DIAMETER) OTHERWISE THE HOLE WILL BE
65 "        ON THE OUTSIDE.
66 "
70 "    4.  ALL PIPE IS TO BE SUPPLIED WITH NOTHING IN THE HOLE
80 "        SO THAT WATER, STEAM OR OTHER STUFF CAN BE PUT INSIDE
90 "        AT A LATER DATE.
95 "
100 "   5.  ALL PIPE SHOULD BE SUPPLIED WITHOUT RUST- THIS CAN BE
110 "       MORE READILY APPLIED AT THE JOB SITE.
115 "
120 "       N.B. SOME VENDORS ARE NOW ABLE TO SUPPLY PRE-RUSTED
130 "       PIPE. IF AVAILABLE IN YOUR AREA, THIS PRODUCT IS
140 "       RECOMMENDED AS IT WILL SAVE A LOT OF TIME AT THE JOB
150 "       SITE.
155 "
160 "   6.  ALL PIPE OVER 500FT(153M) IN LENGTH SHOULD HAVE THE
170 "       WORDS 'LONG PIPE' CLEARLY PAINTED ON EACH END SO THE
180 "       CONTRACTOR WILL KNOW IT IS A LONG PIPE.
185 "
190 "   7.  PIPE OVER 2 MILES(3.2KM) IN LENGTH MUST HAVE THE
200 "       WORDS 'LONG PIPE' PAINTED IN THE MIDDLE SO THE
210 "       CONTRACTOR WILL NOT HAVE TO WALK THE ENTIRE LENGTH
220 "       OF THE PIPE TO DETERMINE WHETHER OR NOT IT IS A LONG
230 "       PIPE.
235 "
240 "   8.  ALL PIPES OVER 6INS.(152MM) IN DIAMETER MUST HAVE THE
250 "       WORDS 'LARGE PIPE' PAINTED ON IT SO THE CONTRACTOR
260 "       WILL NOT MISTAKE IT FOR A SMALL PIPE.
265 "
270 "   9.  FLANGES MUST BE ON ALL PIPES. FLANGES MUST HAVE HOLES
280 "       FOR BOLTS QUITE SEPERATE FROM THE BIG HOLE IN THE
300 "       MIDDLE.
305 "
310 "  10.  WHEN ORDERING 90 DEGREE,45 DEGREE OR 30 DEGREE
320 "       ELBOWS BE SURE TO SPECIFY RIGHT OR LEFT HAND
330 "       OTHERWISE YOU WILL END UP GOING THE WRONG DIRECTION.
335 "
340 "  11.  ALL COUPLINGS SHOULD HAVE EITHER RIGHT OR LEFT HAND
350 "       THREADS, BUT DO NOT MIX THE THREADS- OTHERWISE AS THE
360 "       COUPLING IS BEING SCREWED ON THE PIPE IT IS BEING
370 "       UNSCREWED FROM THE OTHER.
375 "
380 "  12.  BE SURE TO SPECIFY TO YOUR VENDOR WHETHER YOU WANT
390 "       LEVEL, UPHILL OR DOWNHILL PIPE. IF YOU USE DOWNHILL
400 "       PIPE FOR GOING UPHILL YOU WILL END UP GOING THE WRONG
410 "       WAY.
```